情　報　学　基　礎　70－7
デジタル・ドキュメント　38－7
（２００３．３．２８）

文書構造情報の抽出とメタデータ化

川崎洋治[†1]　野村直之[†2]　中川尚[†1]

文書はその本文内部に、自身に関する書誌データ、すなわちメタデータを含んでいることが多い。本稿では、プレーンテキストや HTML 文書の内容記述部分から作者や見出し階層のようなメタデータを自動的に抽出する技術を紹介する。次に、抽出したメタデータを格納するための枠組みを RDF Schema によって定義して、メタデータを流通させる仕組みを提案する。最後に、文書からのメタデータ自動抽出、および Semantic Web 標準に準拠したその構造表現化による情報流通上の効果について論じる。

Automatic Extraction of Document Metadata and its RDF-based Representation

Yoji Kawasaki [†1]　Naoyuki Nomura [†2]　Takashi Nakagawa [†1]

Many documents have their bibliographic information in their body texts, such as creator and chapter-section hierarchy. We have developed an automatic method of extracting this kind of information which is intrinsically embedded in plain-text or HTML documents. Then, the extracted document metadata is restructured into our "RDF Schema"-based representation, which is to be useful in many applications for document distribution and information sharing.

１．はじめに

World Wide Web(WWW)の普及により、インターネットを介して様々な情報に居ながらにしてアクセスできるようになった。しかし一方で WWW は情報の氾濫を引き起こし、求める情報を得るのに次第に手間がかかるようになってきている。現在注目を集めている Semantic Web は、Web 上の様々なコンテンツに機械可読な意味情報（メタデータ）を付与して、Web の有用性を大きく高めようとする狙いがある。しかしながら一方で、情報資源に機械可読なメタデータを一つ一つ手作業で付けるのは、実質的に不可能な作業であり、何らかの支援技術が必要となっている。

情報資源に付与する利用価値の高いメタデータの一つとして、文書構造情報がある。文書構造情報は、文書の話題の大きなまとまりを把握する上で考慮すべき主要な要素[1]であり、文書の章立て構造を残した誤読の可能性が低い要約を作成するのに役立つ[2][3][4]。また、文書の見出し語に現れるタームの重要度を調整することで、情報検索精度を向上させるのにも利用できる[5]。

文書はその本文部分に文書構造情報を含んでいることが多い。本稿ではプレーンテキストや HTML 文書が持っているこのようなメタデータを、文書の内容から自動的に抽出する技術を紹介し、その抽出結果を格納するための語彙定義を RDF Schema[6]で与える。それによって、文書構造情報をメタデータとして流通させる仕組みを提案する。

以下、第２章では、文書内容から構造情報を抽出する技術について紹介する。第３章と第４章では、この抽出した文書構造情報を Web 上で発信するための情報記述の枠組みを提案し、それによる効果について論じる。最後に全体のまとめと今後の課題について述べる。

２．文書構造解析技術

文書内容から構造情報を自動的に抽出する手法として[7]がある。技術系文書とビジネス系文書に対する文書構造テンプレートと、各見出しや作者などの構造要素ごとに数字部、記号部、固有名詞部といった構成パタンを登録し、それらと実際

†１　株式会社ジャストシステム
Justsystem Corporation.
†２　法政大学エクステンションカレッジ
Hosei University Extension College

の文書を照合することによって文書構造要素を判定している。技術文書で９５%、ビジネス文書で８５%の精度で抽出する技術である。しかしルールベースのシステムであるため処理時間の点では決して高速とは言えず、またテンプレートに合致しない場合は判定漏れとなってしまうという問題があった。

他には[2][3]で説明されている要約技術が、プレーンテキストや HTML 文書の内容から構造情報を判定する技術を含んでいる。また[8]では、文書内容から推定した構造情報に基づいてスタイル付けや目次生成する機能の中で文書構造解析技術が利用されている。[9]では、プレーンテキストや HTML 文書の内容から抽出した構造情報を利用して、プレゼンテーションのスライドを生成する「構造化読み込み機能」を提供している。

本章では、この構造解析技術の概要とその解析精度について説明する。また関連する構造抽出技術として、PDF 文書中に存在する表情報を抽出する技術についても紹介する。

２．１．文書構造情報

文書には次のような様々な構造情報が存在する。

・文書タイトルや作成者のような書誌情報
・章立ての階層情報
・べた書き本文部分と図表部分の腑分け
・メール文書の本文、引用、署名の腑分け

この他にも、文書本文に文字列として記述されている日付、電話番号、メールアドレスのような属性情報も一種の構造情報ととらえることができる。本稿で扱う文書構造情報は、主に書誌情報や章立て構造に関する情報とする。

文書の書誌情報を表現するための仕組みには Dublin Core[10]がある。Dublin Core はインターネットにおける情報の発見を目的として開発されてきたメタデータであり、インターネットや電子図書館などにおける標準的なメタデータ規則として広く認められている。Dublin Core では最初、情報資源に付けられた名前を表す Title や、内容を説明する Description などの 15 の要素からなるメタデータ規則が作られた。その後、より詳細で厳密な定義が可能な仕組みの必要性が議論され、Description を TableOfContents(目次)と Abstract(要約)に分けるなど識別子の意味を詳細化したり、値の記述形式を明示するための限定子（qualifier）が定義された[11]。

２．２．文書構造解析

文書構造解析技術で判定する構造情報には大別して、文書内の構造情報自体（以下、文書構造要素と呼ぶ）と、それを助ける補助的情報がある。

２．２．１．文書構造要素

文書内のテキスト文字列を改行コードで区切った行単位で扱うものとすると、次のような意味内容を表す行を文書構造要素と定義する。

・文書主題
　文書の内容を表すタイトル行。HTML 文書の場合は HEAD 内の TITLE タグで指定された文字列を指すこともある。
・章題目
　章節など階層構造を表すための見出し行。
・箇条書き
　同じ性質のものを列挙するとき用いる行。
・日付
　本文中の先頭や末尾に書かれている、作成日付などの文書に関連する日付。
・宛先
　本文中の先頭や末尾に書かれている、文書の宛先。
・作成者
　本文中の先頭や末尾に書かれている、文書の作成者
・引用行
　メール文書における、他のメール内容を引用した行。
・署名ブロック
　メール文書でおもに発信者に関する属性情報が書かれたメール末尾のシグナチャ。

ただし、章題目と箇条書きについては明確に区別する定義がないため、判定時には別の要素とするが精度評価では合算して扱う。図１．に文書例とその文書構造要素を示す。

２．２．２．補助的情報

携帯電話からのメールのような場合を除いて、メール文書では見やすさのために（例えば全角文字３０～４０文字程度で）改行を挿入し、行末を揃えることが多い。メール文書に限らず、このようなタイプの文書をメール型文書と呼び、逆に１段落１改行しか存在しない文書をワープロ型文書と呼ぶことにする。メール型文書の本文はメールとして読むには適度に改行されていて読みやすいが、自然言語処理などの機械処理をしようとすると、段落や文の途中に挿入された改行のために正

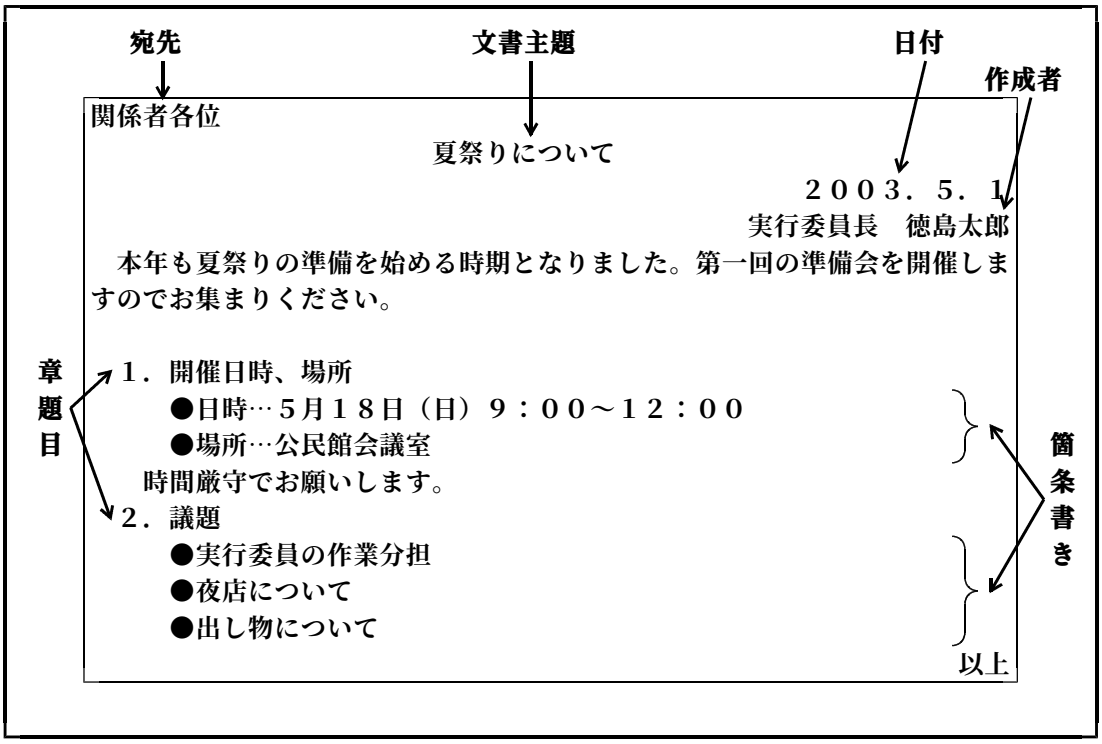

図１．文書構造要素の例

確な処理ができないことが多い。またワープロにメール型文書を読み込んで成形するような場合でも、ワープロ型文書になるように修正するのは手間がかかる。

文書構造解析の際には､文書構造要素に加えて、メール型文書における改行が段落を変えるために必要な改行（必要改行）なのか、行末を揃えるために挿入された不要な改行（不要改行）なのかも判定する。

２．２．３．判定技術

文書構造解析技術は、その性質上様々な局面で必要となる技術であると考えられ、他のモジュールから小さい負荷で呼び出せることが重要である。そのため、形態素解析を使用せずに字面解析だけによって判定することを基本的な技術開発方針とした。手がかりとなる特定の文字列パタンの有無や文書内での行位置、行内の配置というような情報を条件としている。ここで最低限必要となる文字列パタンが数千は存在する。単純な文字列比較で実装すると実用的な速度にはならないため、Aho-Corasick[12]型の文字列照合モジュールを内包している。これによって、理論的には手がかり文字列をさらに増やして判定条件を緻密にしても、判定速度を落とさずに精度向上を目指すことが可能である。

文書構造解析モジュールはファイル形式としてプレーンテキストと HTML を解析対象としている。以下に構造要素ごとに判定条件を概観する。

(1)文書主題

テキスト文書の場合は、インデント量や行頭・行末の空白を除いた実効長、文頭からの行位置、句点が存在しないことなどを条件としている。HTML 文書の場合は、H1 などの意味的構造を表すタグが文書頭の特定の範囲に唯一存在することを第１の条件としている。そのような行が存在しない場合は、フォントサイズや文字飾りの有無、前後の HR タグの有無、センタリングされているか、といった条件をもとに判定する。本文中に文書主題と判定できる行が存在しないときは HEAD 内の TITLE タグに書かれている文字列を採用する。

(2)章立て・箇条書き

テキスト文書では、行頭の文字列パタンの特徴で判定する。HTML 文書では、H2 のような見出しを表すタグが付いているかどうかで判定する。また文書主題と同様に、フォントサイズや文字飾りの有無なども条件に利用する。その後で、同じ行頭形式の行の有無や直後に説明本文が含まれているかといった条件によって、章題目と箇条書きを判別する。

(3)日付
文書の先頭あるいは末尾からの行位置、文字列長、日付らしい文字列パタンの有無といった条件で判定する。
(4)宛先・作成者
文書の先頭あるいは末尾からの行位置、所属名や役職名に特徴的な文字列パタンの有無、さらに宛先では敬称を表す文字列パタンの有無、インデント量や文字列長などを条件としている。
(5)引用行
引用記号としてよく使用される">"や"|"などの文字が行頭またはそれに準じる位置に存在するか、同じ引用記号を用いた行が他にも存在するか、といった条件で判定する。
(6)署名ブロック
文書末尾に"---"のような区切り記号で分断されているか、記号類で囲まれているかというレイアウト的な条件と、署名に記述されることが多い所属名やメールアドレス、電話番号らしい文字列が存在するかといった文字列パタンの条件を組み合わせて判定する。
(7)不要改行
該当行の行末が文末らしいかどうかで判定する。文末らしさは、句点の有無、インデント量、次行が空行や章立てなどの特徴があるか、といった条件を組み合わせて判定する。

２．２．４．判定精度

日本語で記述された次のような文書に対して、文書構造要素を行ごとに振って評価コーパスを構築し、上で説明した方式を実装したモジュールの精度を測定した。すべてのコーパスは判定エンジン洗練のための分析用を兼ねており、ブラインドコーパスによる評価ではない。正否の判断は行単位で独立に評価するため、章題目と箇条書きでは同一形式の行が一貫して判定されているかという観点では評価していない。

・プレーンテキスト
社内に存在したビジネス文書や論説など様々なタイプの１６２９文書から、文書構造上の一貫性や視覚的に構造がわかりやすいかどうかという観点で５段階に分類し、その最も評価が高い３６０文書を採用。

・HTML
様々な Web サイトから取得した１０万を超える HTML 文書から、任意に１０００文書程度を抽出し、その中から構造を持っていて視覚的に構造がわかりやすい３０６文書を選出。Web サイトの HTML であるため、作成者や日付を含む文書は少なく、文書主題と章題目、箇条書き判定を評価するのに使用。

・メール
国内で運営されている複数の技術系メーリングリストに流れた７７５０８メールから、任意に７６９文書を選出し本文のみを抽出。文書主題や作成者､日付を含むメールは少なく、引用行・本文行の腑分けと署名ブロック判定、不要改行判定を評価するのに使用。

(1)テキスト、HTML 文書に対する評価
評価結果を表１．に示す。形態素解析を用いずに字面処理や行内配置情報のみで判定しているため、宛先と作成者の再現率は他の構造要素に比べて低い。また作成者の精度が宛先に対して低いのは、宛先では敬称などの手がかり語が存在することが多いのに対して、作成者では手がかりが少なく判定自体が難しいことによる。
HTML 文書に対する精度は、全般的にテキスト文書に比べて低い。これは、HTML で推奨されている H1 などの意味内容を表すタグを使わずに、行内の配置情報、フォントの大きさや太字などの文字装飾によって、文書主題や章題目を表している文書が圧倒的に多いためである。また逆に、単に太字にしたりインデントするために H1 や UL などのタグを用いている文書も珍しくない。このような HTML 本来の意味通りにタグが使われない文書が多いことが精度低下の一因となっている。

文書構造要素	テキスト文書		HTML 文書	
	適合率	再現率	適合率	再現率
文書主題	94.94%	89.38%	66.35%	66.35%
章題目･箇条書	99.52%	90.89%	91.59%	86.74%
日付	96.92%	97.67%	―	―
宛先	95.52%	72.73%	―	―
作成者	86.05%	63.14%	―	―

表１．テキスト、HTML 文書に対する判定精度

(2)メール文書に対する評価
評価結果を表２．に示す。引用行の判定誤りは、たまたま本文中に挿入されたＣ言語プログラムソースのポインタ記述"->"に過剰反応したのが大きな原因となっている。一方判定漏れは、ごく一部のメールに現在引用記号の条件としている">"、"|"（およびこれらの繰り返し）でなく、":"を使っている場合や、インデントするなど他の方法で引用を表現したものがあったためである。
署名の判定については、表２．では行単位で精

度を計算しているため、全体で一つの意味的なまとまりを表す署名ブロックでは、実際の判定精度を正確には表していない。表３．に署名ブロックごとの精度を算出した結果を示す。署名ブロックの範囲を含めて正確に一致した場合と、ブロックの範囲に重なりがあるものの一部判定漏れや過剰判定がある場合の両方を示す。単に名前だけを書いた１行が末尾に存在するような、判定のための手がかりが少ない場合に判定漏れとなることが多い。また逆に１０行以上にわたるような非常に大きい署名ブロックや、見た目にも本文なのか署名なのかが判別しにくい場合にも判定漏れとなることがある。

文書構造要素	適合率	再現率
引用行	98.93%	99.57%
署名行	98.43%	92.14%

表２．メール文書に対する判定精度

署名ブロックの精度	適合率	再現率
範囲も一致した場合を正解	95.36%	88.12%
範囲に誤りがあっても正解	98.57%	91.09%

表３．メール文書の署名ブロック判定精度

(3)メール文書に対する不要改行判定評価

表４．にメール文書に対する不要改行判定の評価結果を示す。適合率が全般的に低い。これは、メールに含まれるプログラムソース例の記述行を不要改行と判定してしまうことが主な原因である。プログラムソース行は挿入行とみなし、必要改行を正解としている。

不要改行判定精度	適合率	再現率
本文部分	62.92%	90.09%
引用部分	82.67%	91.03%

表４．メール文書に対する不要改行判定精度

２．３．表情報抽出

HTML や PDF 文書が広まるにつれて、内容に表を含む文書が多くなってきている。HTML 文書では表以外の目的に TABLE タグが使われることがあるものの、一旦表であることがわかれば、その表構造を正確に把握することが可能である。一方 PDF 文書では、表の情報は単なる文字列と矩形の塗りつぶし情報としてしか管理されていないため、その表構造を認識するのは一般には困難である。

表情報を抽出する手法には、テンプレートで表構造を定義しておき、マッチした表に対してのみ情報を抽出する手法が考えられるが、いちいちテンプレートを登録する手間がかかる。ここでは、PDF 文書中の座標などのレイアウト情報と文字列情報のみを用いて、表情報を抽出する技術について説明する。

２．３．１．対象とする表形式

文書中に出現する表の例を図２．に列挙する。一般的な表は「罫線がセルを区切る表」であるが、「罫線が省略された表」も存在する。さらに頻度は高くないが、背景色でセルを表現したものや方形でないものもある。ここで説明する表情報抽出技術では、出現頻度が高い「罫線でセルを区切る表」と「罫線が省略された表」だけを対象とする。

【罫線でセルを区切る表】

	電池Ａ	電池Ｂ
電圧	1.5V	2.0V
容量	600mAh	800mAh
価格	150 円	300 円

【罫線が省略された表】

	電池Ａ	電池Ｂ
電圧	1.5V	2.0V
容量	600mAh	800mAh
価格	150 円	300 円

【背景色でセルが表現された表】

	電池Ａ	電池Ｂ
電圧	1.5V	2.0V
容量	600mAh	800mAh
価格	150 円	300 円

図２．表のタイプ

２．３．２．抽出技術

図２．の「罫線でセルを区切る表」が含まれるような文書から、「電池Ａ」の「電圧」が「1.5V」であることを読みとるには以下の情報が必要である。

- どの位置に表が存在するか
- セル構造がどうなっているか
- 項目タイトルを表す行が、どの行・列位置に存在するか（タイトル行・列判定）
- 表の縦と横のどちらの方向が、製品やものの並びで、どちらがそれに付随する属性の並びなのか（表組み判定）

一方 PDF ファイル内には表に関連して以下のように情報が管理されている。

- 文字列のテキストと座標情報

・塗りつぶしなど図形の形状とその座標情報

これらの非常にプリミティブな情報から、表情報を判定する技術について説明する。

(1)表・セル認識

まず PDF 文書から縦長・横長の塗りつぶし領域を罫線候補として取得する。次に「罫線が省略された表」にも対応するため、人間が無意識のうちに補っている罫線をレイアウト的な特徴によって追加する。図２．の「罫線が省略された表」では第１カラムと第２カラムの間、第２カラムと第３カラムの間に縦に一貫した空き領域が存在することを条件に判定する。この後で罫線の交差形状から表を認識しセル構造を判定する。ここまでの条件ではグラフのような図形も表として認識されてしまうため、罫線とセル内文字列の重なり具合や空欄のセルの割合を条件にして、ノイズを除去する。

PDF 文書特有の問題として、太字が同じ文字列の重ね書きで管理されていたり、罫線が長さ方向に細分された多数の連続した罫線群として管理されていることもある。これらは表・セル認識のノイズとなるため事前に正規化する。

(2)タイトル行・列判定方法

左上隅のセル内文字列の有無、セル内文字列の配置方法（中央寄せ、右寄せなど)、文字列の類似度などの条件で判定する。

(3)表組み判定

図２．に例示したような内容の表では、同じ属性の行（または列）には文字列として類似した内容が書かれる場合が多い。この性質を利用して表組みを判定する。例えば 600mAh のセルに着目すると、縦方向よりも横方向の方が文字列一致度合いが高いので、横方向に同じ属性（この場合は容量）が書かれていると判定できる。

２．３．３．判定精度

上で説明した方式を実装し、その判定精度を測定した。Web から収集した表を含むカタログ PDF 文書（69 ファイル、374 表）について正解データを作成し、表情報抽出試作モジュールの実行結果と比較した。セルの判定は、該当セルとその直上セル、直左セルの三つ組みが一致することを正解の条件とした。表組みとタイトル行・列の判定は、表の範囲が正しく判定され、かつそれぞれの項目が判定可能なもののみで評価している。表５．に測定結果を示す。

判定項目		精度
セル認識	再現率	86.72%(12626/14560)
	適合率	88.62%(12626/14248)
表組み	正答率	76.44%(172/225)
タイトル行・列	正答率	84.78%(429/506)

表５．表情報抽出技術評価

【クラス定義】

クラス名	説明	subClassOf
Document	文書を表す。	rdfs:Resource
Divisions	内容項目の集まり。メンバは Division。	rdfs:Seq
Division	内容項目を表す。	rdfs:Resource

【Document を domain とするプロパティ】

プロパティ名	range	subPropertyOf
docTitle	rdfs:Literal	dc:title
creator	rdfs:Literal	dc:creator
date	rdfs:Literal	dc:date
address	rdfs:Literal	――――――
docContents	Divisions	dc:tableOfContents

【Division を domain とするプロパティ】

プロパティ名	range	subPropertyOf
divTitle	rdfs:Literal	dc:title
divContents	Divisions	dc:tableOfContents

表６．クラスとプロパティ

３．RDFによる文書構造の表現

自動抽出した文書構造情報をメタデータとして RDF で記述するために、メタデータの構造を RDF Schema で定義する。まずメタデータとして表現する文書構造要素を次のように定める。

・書誌情報
　文書主題、文書の作成者、日付、
　文書の宛先
・見出しの階層構造情報
　階層構造と各階層ごとの見出し一覧

メール文書の署名ブロック、引用部分との腑分け情報や表情報は、文書または文書の一部を説明する情報ととらえることもできるが、今回は扱わないことにする。

表６．に必要な RDF Schema のクラスとプロパティの一覧を示す。RDF Schema と Dublin Core を表す名前空間をそれぞれ'rdfs'、'dc'で表している。上記の文書構造情報を表現するための起点である文書自身を表すクラス(Document)と、見出しを表すクラス(Division)、各見出し階層ごとに順序付き集合を表すクラス(Divisions)が必要になる。

Document クラスと Division クラスのプロパティのほとんどは Dublin Core のサブプロパティとして定義した。付録に RDF Schema 記述の全体を示す。

図１．に示した文書例のリソースを仮に'http://jichi.org/doc1/'としたときの、このリソースに対する RDF のグラフは図３．のようになる。ここで RDF Model and Syntax[13]に対応する名前空間を'rdf'、本稿で定義した RDF Schema 記述に対応する名前空間名を'ds'とする。

４．メタデータ化における効果

文書構造解析結果のメタデータ化において、様々な効果が期待できる。まず、文書構造が文書内容から推定して自動抽出できることで、メタデータ付与を効率化することができる。構造解析精度が 100%というわけではないので完全自動化はできないが、修正のためのインタフェースを工夫することで、一から付与するよりは遙かに短時間で、メタデータ化可能である。これによって既存文書へのメタデータ付与が促進される。

メタデータの情報流通上の効果としては、まず情報検索における高機能化、高精度化があげられる。文書に付与されたメタデータをキーにして、よりきめ細かい情報検索が可能になる。「○○さんが□□頃に作成した△△という内容の文書」といった検索ができるようになる。また情報検索用のインデックス作成時に、見出しなどのメタデータとして付与されている情報を重要視するチューニングをすることで、より検索者の意図に合致する検索をすることも可能になる。

別の効果としては、文書やデータベース作成時に他情報源のメタデータを利用して情報価値を高めることがあげられる。例えば、書誌情報データベースを自動生成するという利用ができる[14]。また商品カタログや観光案内などのように他の情報源から文書を自動生成するような場合でも、基となる情報源のメタデータを埋め込むような仕組みを実現することが可能になる。

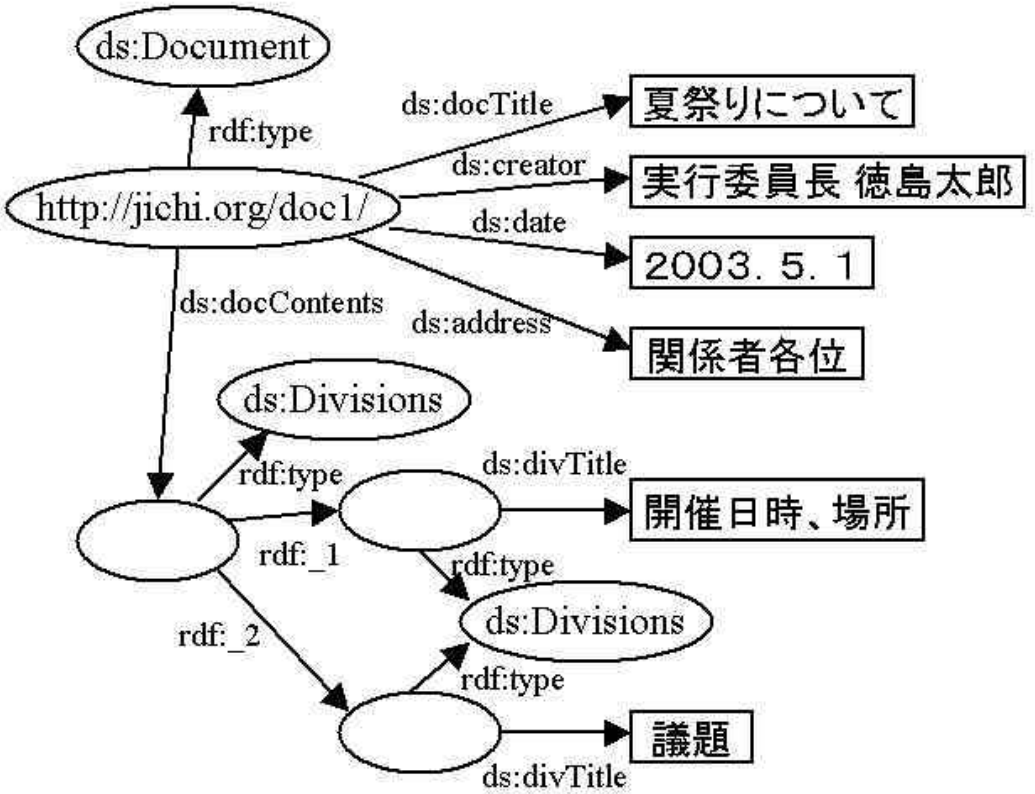

図３．RDF のグラフ

５．まとめ

本稿では、文書構造情報を自動判定する技術について説明し、その構造情報をメタデータとして記述するための枠組みを RDF Schema によって定義した。既存文書から文書構造情報を自動判定することでメタデータ化が促進されるほか、文書構造情報をメタデータとしてアクセス可能にすることで、情報検索の高機能化・高精度化の実現、他情報源のメタデータを利用することによる文書やデータベースの自動生成、といったその効果についても論じた。今後は、メールの引用・署名ブロックの情報や表情報についても、メタデータ化していくという展開が考えられる。

【参考文献】

[1] 仲尾由雄，話題の階層構成に基づく文書自動要約：本一冊を一頁に要約する試み，情報処理学会研究報告 NL-132-7, pp.49-56(1999).

[2] 野村直之, ConceptBase の言語処理と新しいソリューション，情報処理学会研究報告 NL-129-1, pp.1-8(1999).

[3] 野村直之，ナレッジマネジメントツールの配備、実践動向と次世代技術，人工知能学会誌 Vol.16 No.1, pp.33-41(2001).

[4] http://www.justsystem.co.jp/km/product/cb104.html

[5] http://www.namazu.org/

[6] http://www.w3.org/TR/rdf-schema/

[7] 土井美和子、福井美佳、山口浩司、竹林洋一、岩井勇，文書構造抽出技法の開発，電子情報通信学会論文誌，Vol. J76-D-II No.9, pp. 2042-2052(1993).

[8] http://www.ichitaro.com/13/word/reason5.html

[9] http://www.justsystem.co.jp/justarks/prezark/index.html

[10] http://dublincore.org/

[11] http://dublincore.org/documents/dcmi-terms/

[12] Aho, A. V. and Corasick, M. J., "Efficient String Matching : An Aid to Bibliographic Search", Comm. ACM, Vol.18, No.6, pp.333-340.

[13] http://www.w3.org/TR/REC-rdf-syntax/

[14] 野村直之、小林茂，セマンティック Web の動向とメタデータ，第４回 XML コンソーシアム Day 発表資料(2003).

【付録】文書構造情報のRDF Schema記述

```
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf      'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY rdfs     'http://www.w3.org/2000/01/rdf-schema#'>
    <!ENTITY dc       'http://purl.org/dc/elements/1.1/'>
    <!ENTITY dcterm 'http://purl.org/dc/terms/'>
    <!ENTITY ds       'http://www.justsystem.co.jp/mary/la-ns#'>
 ]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:ds="&ds;"
          xmlns:dc="&dc;" xmlns:dcterm="&dcterm;">

  <rdfs:Class rdf:ID="Document">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Division">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Divisions">
    <rdfs:subClassOf rdf:resource="&rdfs;Seq"/>
    <rdfs:member rdf:resource="&ds;Division"/>
  </rfds:Class>

  <rdf:Property rdf:about="&ds;docTitle">
    <rdfs:domain rdf:resource="&ds;Document"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:subPropertyOf rdf:resource="&dc;title">
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;creator">
    <rdfs:domain rdf:resource="&ds;Document"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:subPropertyOf rdf:resource="&dc;creator">
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;date">
    <rdfs:domain rdf:resource="&ds;Document"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:subPropertyOf rdf:resource="&dc;date">
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;address">
    <rdfs:domain rdf:resource="&ds;Document"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;docContents">
    <rdfs:domain rdf:resource="&ds;Document"/>
    <rdfs:range rdf:resource="&rdfs;Divisions"/>
    <rdfs:subPropertyOf rdf:resource="&dcterm;tableOfContents">
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;divTitle">
    <rdfs:domain rdf:resource="&ds;Division"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
    <rdfs:subPropertyOf rdf:resource="&dc;title">
  </rdfs:Class>

  <rdf:Property rdf:about="&ds;divContents">
    <rdfs:domain rdf:resource="&ds;Division"/>
    <rdfs:range rdf:resource="&rdfs;Divisions"/>
    <rdfs:subPropertyOf rdf:resource="&dcterm;tableOfContents">
  </rdfs:Class>

</rdf:RDF>
```